

## Block 2, Part 6: Client server interaction: A basic web application

---

*Peter Thomson (2017), Dave McIntyre (2019) and Stephen Rice (2021, 2022)*

### 1 Introduction

We have seen how data sent from a web form over a network can be stored in a database on a server. It can be extracted and manipulated, and using PHP it can be used to add content to a web page.

In this part we will explore how database data can be edited and updated through a web page interface.

We will also further investigate ways to validate data arriving from a web form, and further enhance the security of the server and the data that it stores in the database.

### 2 Learning outcomes

On successfully completing this part of Block 2 you should be able to:

- explain the need for planning in any software development
- create PHP and HTML pages to validate data on the server and to select, edit and update data records
- reuse code for a new website
- discuss security issues for HTML pages that use PHP and a database.

### 3 Web application structure

Our example application becomes more complex as the interaction increases. When you create web applications it is your initial planning that organises the code and keeps everything in its place. When you study web applications it helps to look for aspects that illustrate the overall plan, and provide a framework to organise the detail.

When creating a web application you need to understand each code statement, what it does and where it fits in the context of the overall application. This means reading the code itself, taking note of the function names from the programming language and how the parameters are being passed to that function. A key test of your understanding of the code is being able to use and modify code to apply it to a new situation.

## Row identity

To maintain data in a database table through a web page interface, we first need to be able to search for and then select the row of data we are interested in. Having identified a particular row, we then need to be able to use our web page interface to edit and update the data it contains. In our application, the identity of the row of interest is placed in a hidden form input named 'id' and made available in the application in the variable `$id`.

Hidden form elements are a very useful way of passing information between the client and the server without displaying anything to the user. We will be using hidden form elements again in a later section. However, it is essential to be aware that they are only hidden from the page displayed in the browser. If the user views the source code, which we cannot prevent then they can view (and potentially change) the hidden data.

## Validation

Another task required on the server is to validate data from any web page form as it is received. We've seen that validation on the client side, using HTML and JavaScript, helps the user complete a form correctly, but the presence of client side validation doesn't remove the need to further validate on the server. If the data is found not to be valid when it arrives at the server, appropriate action needs to be taken. Here we have decided that it needs to be returned to the user inside the same HTML form, with a warning that indicates the invalid data that needs to be changed. That will not be the same in every application.

Both sending the data to the web client to edit, and sending the data to the web client to change because it isn't valid, use the same techniques to return data already inside an HTML form to the client. We don't want to have multiple copies of the same HTML form as this makes software maintenance much more difficult. Instead we will modify our process of creating the form so that the same single form is used each time.

## Flags

A 'flag' is a name given to a variable that is used to signal which choice has been made or to record the current state of some operation within some software. Sometimes we refer to this as 'setting' a flag.

In this section we use several flags. One is being signalled by a hidden web form element named 'task'. This flag is being used to signal the purpose of the form data is submitted to the server. The flag named 'task' could take on many different values. For example, we might use a value of 'delete' to signal that the row identified by the 'id' value should be deleted. This flag is passed by using submit inputs named 'task' with the value of the task to carry out.

In the following example a submit button named 'task' is given a value of 'save' to indicate that the form data is intended for saving on the server, and needs to be validated, and if valid then passed to the save routine to add a new row or update an existing row identified by the 'id' value.

```
<input type="submit" name="task" id="save" value="save" />
```

This value is made available in the application in the variable `$task`.

Another flag we are using is the variable `$valid`. This is given a boolean value and is initially set:  
`$valid=true;`

In PHP, if you echo a boolean value to help you debug your code, the Boolean **true** is converted to a string "1" but the Boolean false is converted to an empty string "". It is better to use `var_export()` for debugging rather than `echo()` as it handles all PHP data types.

If the server side validation detects that any item of the form data is not valid, the PHP code changes `$valid` to false; a message is created to inform the user of this problem, and the form, along with its data and validation feedback are returned to the client.

The core PHP file: **b\_admin.php** uses these flags to determine which files are required, so that any processes that are not appropriate at that point are not loaded or displayed. For example, the process to validate form data is only needed if form data has been sent to the server to be saved, and the processes to save data to the database are only needed if that form data is valid. The extract below shows the initial values of flags and other variables used by different components of the application.

```
$url = $_SERVER["PHP_SELF"]; // URL of this page for forms to POST to
$columns = []; // list of names of columns in database table
$task = ''; // task to carry out in response to form submission
$id = ''; // ID of row in table being viewed/edited (if any)
// Search criteria used by search form and data table
$search = ''; // value to search for
$sort = 'id'; // column to sort by $order = 'ASC'; // order to sort by
// Data shown on data entry form
$data = []; // key/value data from form submission or row in database table
$valid = true; // whether data in $data is known to be valid
$feedback = []; // key/value feedback about invalid data
```

## Order of operations

The application starts by requiring code to output the head of the page, connect to the database, read columns from the database table, read submitted form data into `$data`, `$task` and `$id` as shown in the following extract:

```
// Connect to the database
require "connect.php";
// Read column names from the database table into $columns
require "b_read-columns.php";
// Read the submitted form data into $data
require 'b_read-post.php';
// Read $task and $id (if any) from submitted form data
if (!empty($data['task'])) {
    $task = $data['task'];
}
if (!empty($data['id'])) {
    $id = $data['id'];
}
```

The 'task' and 'valid' flags are then used to decide which tasks to carry out using the row 'id' and form 'data', as shown in the following extract:

```

// Task is to start editing a row...
if ($task == 'edit') {
    // So read the row from the database table into $data
    // This data will be presented by the data form
    require "b_read-row.php";
}

// Task is to save submitted form data to a row in the database table...
if ($task == 'save') {
    // So validate the form data
    require "b_validate.php";
    if ($valid) {
        // And save the row (this will clear $data and $id)
        require "b_save-row.php";
        // Row has been saved so reset task/data/id
        $task = '';
        $data = [];
        $id = '';
    }
}

// Task is to delete a row from the database table...
if ($task == 'delete') {
    // So delete the row (this will clear $id)
    require "b_delete-row.php";
    // Row has been saved so reset task/data/id
    $task = '';
    $id = '';
}

```

Finally, the application requires code to output a search form and process search form data, to output a table of rows from the database table, and to output a form to add/edit/re-enter data to a row in the database table, as shown in the following extract:

```

// Output a form to search or sort with
// This also processes search criteria used by the data table
require "b_search-form.php";
// Output a table of rows in the database table matching search criteria (if any)
require "b_data-table.php";
// Output a form to enter or edit data (using $data and $id)
require "b_data-form.php";

```

## Activity 1 Observe the operation of code designed to search, edit and update data in a database

 Allow 60 minutes for this activity

Unzip all the files from the Block 2 Part 6 downloads and then upload them to your TT284 server at <https://oucu.tt284.open.ac.uk/>.

It is best to remove the files from Part 5 first, but you should retain on the server the file **credentials.php** containing your existing server user credentials including your database name, username and password to avoid having to enter them again.

If you wish to keep the Part 5 files you can download them to your local machine before removing them from the server.

From the TT284 server open **b\_admin.php** in your web browser.

Now examine the code in this file in your text editor as you use this page in your browser to carry out a set of operations to search for and then edit a row of data. Each row of data is one record in our database.

The content of the page is split into a number of different sections that are revealed when you click each Show/Hide button.

Look at the messages displayed in the browser that show if the different PHP scripts have been executed and what they found. This is revealed by the 'Show/Hide: Script execution reports' button.

Observe how the reports and the values of the flags change as you carry out each of the different operations: to add a new record, to correct a record that doesn't validate, to search for and edit a record, and to delete a record.

## 4 Editing existing data

Previously, we created PHP code that could be used to search and sort data from the database.

### Add row identity and task buttons

We extend this routine to provide 'edit' and 'delete' buttons next to each item that is displayed.

The HTML form to search for an item to edit is generated by the file **b\_search-form.php**. This is revealed by the 'Show Hide: Search Form' button.

The rows that match this search are then returned in a HTML table generated by the file **b\_data-table.php**.

To make each row in this table of matches editable we have added a form for each row in the final column. This contains a hidden input with the row id, and two submit buttons to set the task – one to edit the row with the hidden id, one to delete the row:

```
<form method="POST" action="<?php _e($url); ?>">
  <input type="hidden" name="id" value="<?php _e($row, 'id') ?>" />
  <input type="submit" name="task" value="edit" />
  <input type="submit" name="task" value="delete" />
</form>
```

The PHP script must next output the form that the user can use to edit the row that has been selected.

## Read a row from the database table

When we wish to edit some data that already exists in a database, we need to retrieve it from a database and then present the data within a form. The retrieval is done in the file **b\_read-row.php** which is executed using 'require' when the task is 'edit', as follows:

```
$sql = "SELECT * FROM $database_table WHERE id=?";
// Prepare statement using SQL command
if (!$stmt = $database->prepare($sql)) {
    die("Error preparing statement ($sql): $database->error");
}
// Bind one string ('s') parameter:
// Replace the first '?' in $sql with the value in $id
if (!$stmt->bind_param('s', $id)) {
    die("Error binding statement ($sql): $stmt->error");
}
// Execute statement and get result
if ($stmt->execute()) {
    $result = $stmt->get_result();
} else {
    die("Error executing statement ($sql): $stmt->error");
}
// Read the first row as an array indexed by column names
$data = $result->fetch_assoc();
```

The `fetch_assoc()` method retrieves a row of data into an associative array. That means that the key of each value in the array will reflect the name of the column in the table for each field in the row, just like the data submitted by the data entry form.

## Add row data and identity to the data form

The row data is then output as the value for each form element in the form output by `b_data-form.php`. As the data in the database and data submitted by a form may potentially contain malicious content, the `htmlspecialchars` function, via a 'helper' function called `'_e'` is applied to each data item before it is added to the web page returned to the user, as follows:

```
<label for="email">Email:</label>
<input
    type="text"
    id="email"
    name="email"
    value="<?php _e($data, "email") ?>"
    onchange="validate(event.target)"
/>
```

A hidden input at the start of the form is used to record the identity of the record being edited, so that when the form is submitted, it updates the existing row:

```
<input type="hidden" name="id" value="<?php _e($id) ?>" />
```

This form is then output as part of the web page in response to the user's submission to the server.

## Validate form data

The user can now edit the data and submit the form. The JavaScript provides the same validation as an HTML email input – it checks that a '@' symbol is present, but not that it is followed by a valid domain name. The JavaScript (or HTML) validation could also be bypassed, so simply making this validation stricter would not ensure security.

Let us edit the email, but introduce a deliberate error in the email as though we mistyped it, typing e.g. 'examplecom' rather than 'example.com' after the '@' symbol. If we submit the form, what we now see is the same form, but with clear messages that the server has found an error in the submission, and the form also contains the data that we submitted. This app is using this method of displaying the messages to help you understand the processes.

The data we submitted from the edit form was extracted from \$\_POST. Because the submitted data also contains the 'task' flag set to 'save'; the data is passed to the validation routine on the server by requiring **b\_validate.php**.

We will look at this file in more detail in a later section.

In our example, firstname and lastname don't cause any problems with validation, but the email doesn't validate as an email address without a valid domain name.

Note that because some requirements such as those to validate an email are very common, there are built-in filter functions in PHP which allow us to do those common tasks without writing regular expressions.

The 'FILTER\_VALIDATE\_EMAIL' constant can be used with the 'filter\_var' function to give a boolean true if the email matches a valid pattern or false if it does not.

```
$value = $data['email'];  
// If value does NOT match the filter then it is invalid  
if (!filter_var($value, FILTER_VALIDATE_EMAIL)) {  
    $feedback['email'] = 'Server feedback: Onlye valid email addresses are permitted';  
    $valid = false;  
}
```

## Add server feedback to the edit form

Our flag \$valid is now false so the data is not saved, and the PHP process continues to the script **b\_data-form.php** that we described earlier, except that the data in \$data is from the form submission rather than from the database. Additionally, the feedback message from \$feedback['email'] is output in the span element with id 'feedback\_email' after the email input. This is the same element used to present client-side validation feedback.

```
<span id="feedback_email" class="invalid">
    <?php _e($feedback, "email") ?>
</span>
```

If the data in the form is now edited back to a valid email, and submitted, it will again be checked by the validation routine on the server, and if valid will be saved.

## Save form data to the database

When an existing row is saved, it must be saved back as the same row – the database row must be updated. We are using an auto-generated id column to guarantee that the id is unique for each row. The **b\_save-row.php** file checks if the `$id` variable has a value and chooses between an SQL INSERT or UPDATE statement accordingly:

```
if ($id) {
    // Update existing row
    $sql = "UPDATE $database_table SET firstname=?, lastname=?, email=? WHERE id=?";
} else {
    // Create new row
    $sql = "INSERT INTO $database_table (firstname, lastname, email) VALUES (?, ?, ?)";
}
```

This SQL is used to create a prepared statement which is bound to the form data and id value of the row to update, and executed.

### Activity 2 Observe and track the validation process from form to server and back to form again

 Allow 60 minutes for this activity

Follow the process of validating an item first on the client and then on the server by comparing what you see on the client as you add some data to the form, then as you submit, and then through the messages that are returned by the server.

Start with brief invalid data. The JavaScript has been set to accept any number of letters, spaces and digits for the first and last name, but the server will reject digits and impose a minimum/maximum length. The email validation is stricter on the server as described earlier.

Examine the data being sent to and from the form each time. Try to track the process in the code.

Observe the changes that are made to the record.



## 5 Validating data on the server

Previously we received the data from the form, and then stored it in the database. Now we are going to look in more detail at the validation step inserted between those two processes on the server. Both the form for new data and for edited data will set the action to 'save', and this is what `b_admin.php` checks for before requiring `b_validate-data.php`.

### Add a validation step before saving and clearing form data

The `b_admin.php` sets `$valid` to true so that data may be saved, but if it is found to be invalid, `$valid` will be set to false in `b_validate-data.php`.

If `$valid` is false, then the data is not saved as the `b_save-row.php` file is not required and not executed, and the invalid data and feedback is presented on the data entry form.

If `$valid` is true, that is no errors have been found, then the `b_save-row.php` file is required and executed, and the data array is cleared so the data entry form is ready to enter another row.

```
// Task is to save submitted form data to a row in the database table...
if ($task == 'save') {
    // So validate the form data
    require "b_validate.php";
    if ($valid) {
        // And save the row (this will clear $data and $id)
        require "b_save-row.php";
        // Row has been saved so reset task/data/id
        // so that the data table is shown again and the data form is empty
        $task = '';
        $data = [];
        $id = '';
    }
}
```

### Validate each field and provide feedback if invalid

Usually you would try and set the JavaScript validation to match the validation on the server, but here we want there to be some differences so that you can see the server side validation in action.

The following code is taken from the required file `b_validatedata.php` (note that it won't run on its own).

The 'preg\_match' function uses a regular expression to check a match. As a demonstration here we are checking 'firstname' and 'lastname' data for a match to only letters of the alphabet in upper or lower case, and a space character. We can have between 1 and 30 of these characters. In practice people's names can use a huge range of characters in a very diverse range of alphabets.

Regular expressions in PHP work much the same as the ones we already saw in JavaScript in Part 3, though note that in PHP regular expressions are strings not objects, so must be enclosed in single or double quotes. The 'preg\_match' function provides similar functionality to the regex 'test' method in JavaScript, in that it

returns a truthy value if it finds a match and a false value if not. We pass two arguments, the regular expression, and the value to check against the expression. We use the '!' operator to invert the result of calling 'preg\_match' as we only need to provide feedback if there is no match.


```
// Assume data is valid until we find an error
$valid = true;
// Start with no feedback
$feedback = [];
$value = $data['firstname'];
// ^$ = anchors, [a-zA-Z ] = letters/spaces, {1,30} = 1-30 characters
$format = "/^[a-zA-Z ]{1,30}$/";
// If value does NOT match the format then it is invalid
if (!preg_match($format, $value)) {
    $feedback['firstname'] = 'Server feedback: Only 1-30 letters/spaces are permitted';
    $valid = false;
}
```

If the comparison is valid we do nothing, but if the comparison is not valid we ensure `$valid` is false and create a `$feedback[]` array item that contains advice to the user which is included in the data form.

## 6 Creating new sites by reusing code

Creating reusable code and making use of existing code speeds up the development process, and if you reuse the right code, ensures that you don't make a lot of mistakes in code development. You should, however, always make sure that you understand why code is written in a particular way before you reuse it.

### Activity 3 Checking how well you understand the application

 Allow 120 minutes for this activity

Your task here is to add a field for a telephone number to the application, and to validate it in JavaScript and PHP.

This may be a lot less work than you expect, as we have made the code as reusable as possible.

This is the most important exercise before the TMA. It is worth spending some time on this.

Create a new folder on your computer for this project. Into this folder, copy all the files that start **b\_** from the Part 6 folder, plus **credentials.php**, **connect.php**, **helpers.php**, **table-manager.php** and **style.css** from the Part 5 folder.

Rename each file where the file name starts with the characters **b\_** so that the name starts with the characters **c\_**.

Next, use find and replace to change all occurrences of 'b\_' in `c_admin.php` to 'c\_'.

### Create database table

Next create a new database table called 'test\_guests' that has the same 'id', 'firstname', 'lastname', and 'email' fields as 'tt284\_guests' plus an additional field named 'telephone' which will store a string value. Edit the relevant lines in **table-manager.php**:

```
// TODO: Change this value to configure the application
$database_table = "tt284_guests"; // name of database table to create/drop
// SQL to create table with appropriate fields
// TODO: Change these columns to meet application requirements
// There must be a column between each field BUT no comma after the last field
$create_sql = "CREATE TABLE $database_table (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(30) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    email VARCHAR(50) NOT NULL
)";
```

Change 'tt284\_guests' to 'test\_guests' and after the 'email' column, add a comma and the additional column for telephone. You can use the type VARCHAR(12) for this field.

Save this file, then upload it to the server and run it. Click 'create' to create the table or 'drop' if you need to remove it to change its structure.

Edit the file **c\_admin.php** to change the value of \$database\_table to that of the table you have just created.

Now upload all the relevant files in this folder to the server and then open c\_admin.php in your web browser. If you have uploaded everything required, it should run without error.

Now look at the display of each section on the web page – the search form should show the new telephone field as we have generated this from the database, but the data entry form only shows the original form fields.

## Add a field to the data entry form

Edit the file **c\_data-form.php**.

Look for the lines which create the input element for the email field:

```
<p>
    <label for="email">Email:</label>
    <input
        type="text"
        id="email"
        name="email"
        value="<?php _e($data, "email") ?>"
        onchange="validate(event.target)"
    />
    <span id="feedback_email" class="invalid">
```

```
<?php _e($feedback, "email") ?>
</span>
</p>
```

Copy these lines and paste them below the email field. Now edit them to be the input element for telephone by changing each occurrence of 'email' to 'telephone'.

Save this file and upload it to the server and rerun c\_admin.php by reloading the page in your web browser.

Now the data entry form should show four fields that include telephone although when you add and edit records, the telephone number is not saved.

You may see an alert from the client-side validation script that the telephone input is unknown, we will fix this later. If you see a database error that the telephone field has no default value, this is due to appending 'NOT NULL' to the column type when you created it – you can go back and recreate the table without this restriction or continue to complete the application.

## Add a column to the data table

Next, edit the file **c\_data-table.php**.

Add 'telephone' to the SQL query SELECT and WHERE components so that it becomes:

```
$sql = "SELECT id, firstname, lastname, email, telephone";
$sql = $sql . " FROM $database_table";
$sql = $sql . " WHERE firstname LIKE ? OR lastname LIKE ? OR email LIKE ? OR telephone LIKE ?";
$sql = $sql . " ORDER BY $sort $order";
```

Also change the lines that bind the prepared statement used to search the database table to expect an additional parameter:

```
if (!$stmt->bind_param('ssss', $term, $term, $term, $term)) {
    die("Error binding statement ($sql): $stmt->error");
}
```

Next, add a line for 'telephone' after the lines that output table headings and table data, respectively. To keep the final row of the table in line, increase the 'colspan' (column span) attribute value from 2 to 3:

```
<th>telephone</th>
<td><?php _e($row, 'telephone') ?></td>
<td colspan="3">
```

Again, save this file and upload it to the server, then run c\_admin.php again. You should now be able to search and sort table data and see the additional heading for 'telephone', although you will not see any data in this column as it is not yet being saved.

## Add a column to the save data

Now edit **c\_save-row.php**.

We need to add 'telephone=?' to the UPDATE line, and 'telephone' and a fourth '?' character to the INSERT line:

```
if ($id) {
    // Update existing row
    $sql = "UPDATE $database_table SET firstname=?, lastname=?, email=?, telephone=? WHERE id=?";
} else {
    // Create new row
    $sql = "INSERT INTO $database_table (firstname, lastname, email, telephone) VALUES (?, ?, ?, ?)";
}
```

Then a few lines down add an additional 's' to each of the lines that bind the prepared statement, and add the telephone data to arguments list after email (but before id):

```
if ($id) {
    // Bind parameters for UPDATE statement ('s' for each column plus 's' for id)
    if (!$stmt->bind_param('sssss', $data['firstname'], $data['lastname'], $data['email'], $data['telephone'], $id)) {
        die("Error binding statement ($sql): " . $stmt->error);
    }
} else {
    // Bind parameters for INSERT statement ('s' for each column)
    if (!$stmt->bind_param('sssss', $data['firstname'], $data['lastname'], $data['email'], $data['telephone'], $id)) {
        die("Error binding statement ($sql): " . $stmt->error);
    }
}
```

Again, save this file and upload it to the server, then run c\_admin.php.

Now add several rows of data to the server. Check that the data is being saved and that you can add, edit and delete rows. Observe the telephone number is not yet validated.

## Client-side validation

Now all that is left to do is to update the validation. For the client-side validation, edit **c\_admin.js**. This contains two functions: 'validate' and 'validateForm'. First add the telephone field to the 'validate' function. Copy the code that sets a validation pattern and feedback message for the 'firstname' field and paste it after the code for the 'email' field. Change the first line to check for the id 'telephone' and change the regular expression pattern to accept only numbers and spaces `/^[0-9 ]*$/` and change the feedback message accordingly.

```
if (inputElement.id == "telephone") {
    // ^$ = anchors, [a-zA-Z0-9 ] = letters/digits/spaces, * = 0 or more characters
    pattern = /^[a-zA-Z0-9 ]*$/;
    feedback = "Only letters, numbers and spaces are permitted";
}
```

Then add a line for the "telephone" field to the 'validateForm' function by copying, pasting and editing an existing line:

```
valid = valid && validate(document.getElementById("telephone"));
```

Save this file and upload it to the server.

Now run `c_admin.php` and add a range of valid and invalid data to see how the validation performs. It should restrict what you can type in the telephone field but not the length, which may be too short or too long. To provide security, we will add a stricter check in the server-side validation.

## Server-side validation

Edit **`c_validate.php`** and locate the lines that validate existing fields. Add the following lines after the email field validation:

```
$value = $data['telephone'];  
// ^$ = anchors, [0-9] = digits, {8,12} = 8-12 digits  
$format = "/^[0-9]{8,12}$/";  
// If value does NOT match the format then it is invalid  
if (!preg_match($format, $value)) {  
    $feedback['telephone'] = 'Server feedback: Only 8-12 digits allowed';  
    $valid = false;  
}
```

Save this file and upload it to the server and reload `c_admin.php` to download the new JavaScript to your web browser.

Now if you enter a telephone number less than 8 or more than 12 digits, the client-side validation will allow it but the server-side validation should not. In reality some telephone numbers are more complex so a real application would have a more complex format but 8 to 12 digits works well enough for this example.

Congratulations! You have successfully adapted this web application to meet a new requirement.

Discuss your observations and your code in the Block 2 Forum.

As you move on to other modules and develop your programming skills you will meet other styles of writing code that are even more reusable than those in this block, and encounter frameworks that make it easier to build complex applications without repeating yourself.

## 7 Security of web applications

One of the important themes that run through this module is the need for security concepts to be applied at all stages of design, development and implementation. Security must be designed in detail right from the start for any web application. It cannot easily be added later.

The web application in this part of the block includes the following security measures:

- PHP files that we require or include should not run on their own.
- Data from a form should never be used directly. It should always go through an appropriate escaping process before output in HTML to prevent cross-site scripting.
- Data from a form should always be validated on a server before it is used or stored.
- Prepared statements should always be used if any data from a form is used in an SQL statement to prevent SQL injection

There are several other aspects that we have started to consider and will explore in more depth later in the module:

- Before an application can be used in the real world, all communication between server and client must take place using HTTPS, to keep data secure in transit.
- The user must be securely authenticated (logged in) and perhaps logged out if inactive for a period of time.
- The user must also be authorised to perform the operations they request.
- Passwords should not be stored as plain text. They should be stored using encryption such that passwords can be checked but not decrypted.
- The server itself and the database should be fully secured. All data should be backed up and kept securely.
- The app developer needs to be aware of the many ways in which the users of a website can be attacked. Cross-Site Request Forgery (CSRF) is one example.
- There is also a lot of detailed legislation that covers what data you are allowed to store, where and how you are allowed to store it and the security that must be implemented.

The developer needs to be aware of the risks from many different directions!

## Where next?

We have completed a series of demonstrations that illustrate how HTML, CSS, JavaScript and PHP can work together to create web pages that use, modify and validate data in a database. These concepts now need to be applied to the TMA.

In Block 3 we will explore more aspects of web application development:

- How web page content and functionality can adapt to the device it is presented on, particularly mobile devices with smaller screens, no keyboard or mouse, but a range of other input methods and sensors
  - How web applications can communicate more efficiently between client and server, requesting only the data needed and updating the page without reloading.
  - How web applications can maintain their state on the client and on the server between requests, or even work offline.
-